Basic Power Analysis with Logistic Regression

Data Science Implementation

2025-02-17

A two-sample test is a statistical test used to check whether two independent datasets come from the same probability distribution.

Let's define our datasets:

$$\text{Let } X = \{X_1, X_2, ..., X_n\} \sim F_X, \quad Y = \{Y_1, Y_2, ..., Y_m\} \sim F_Y$$

We test the hypotheses:

$$H_0 : F_X = F_Y \quad \text{(The two distributions are the same)}$$

$$H_1 : F_X \neq F_Y \quad \text{(The two distributions are different)}$$

Dataset choice

We need a dataset that: - Simulates real-world variability: The data should contain structured differences between two populations while maintaining some overlap. - Allows for easy visualization and interpretation: Since we're exploring two-sample testing, we want a dataset with two features that can be plotted in 2D. - Introduces controlled differences between the two distributions: By setting different mean vectors and covariance structures, we ensure that the two groups are similar but not identical. Our goal is to see how well a classifier can distinguish between the two groups.

```
set.seed(22) # my lucky number
n_samples <- 500  # Number of Data points per group, not too many, not too little
n_features <- 2   # Number of dimensions(aka features, variables)
```

A multivariate normal distribution is a generalization of the normal distribution to multiple dimensions. It is defined as:

$$X \sim N(\mu_X, \Sigma_X), \quad Y \sim N(\mu_Y, \Sigma_Y)$$

where: - $\mu' sare the mean vectors (determining the center of the distributions). - \sim' sare the covariance matrices (determining the sha$

The covariance matrix influences the shape of the data cloud:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$

where: - $\sigma\_12\&\sigma\_22\&are variances of the two features. - \rho is the correlation coefficient.$

```
# means vectors for X and Y (where both distribution are centered) on cartesian map
mu_X <- c(2, 2)
mu_Y <- c(0, 0)
# covariance matrices
Sigma_X <- matrix(c(1, 0.5, 0.5, 1), nrow=2)  # Correlated variables for X
Sigma_Y <- matrix(c(1, -0.3, -0.3, 1), nrow=2)  # Slightly negatively correlated for Y
# ensure different internal structures
```

```
library("MASS")
#?mvrnorm
# multivariate normal samples
X = mvrnorm(n_samples, mu_X, Sigma_X)
```

```r
Y = mvrnorm(n_samples, mu_Y, Sigma_Y)

df_X = as.data.frame(X)
df_Y = as.data.frame(Y)
df_X$group = "X"
df_Y$group = "Y"
df = rbind(df_X, df_Y)
colnames(df) = c("Feature1", "Feature2", "Group")
df$Group = as.factor(df$Group)
head(df)
```

```
  Feature1 Feature2 Group
1 1.043625 2.069324     X
2 5.034089 3.270376     X
3 2.951258 2.794348     X
4 2.522732 1.984438     X
5 1.193057 2.445015     X
6 3.769387 3.448924     X
```

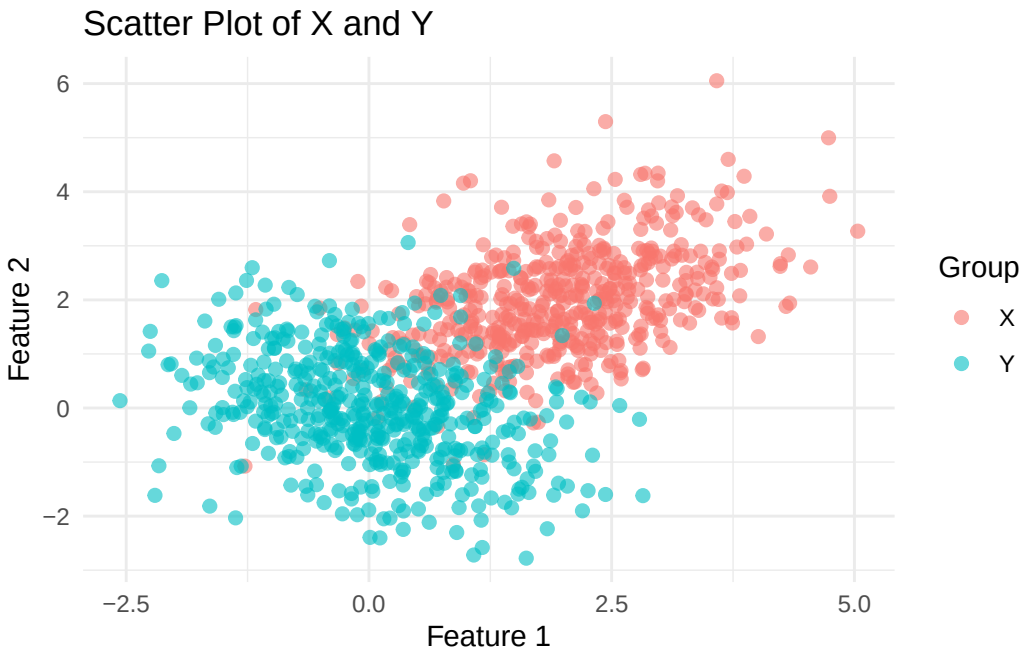
```r
summary(df)
```

```
    Feature1           Feature2        Group
 Min.   :-2.56366   Min.   :-2.77769   X:500
 1st Qu.:-0.06173   1st Qu.:-0.07903   Y:500
 Median : 0.96118   Median : 1.04559
 Mean   : 0.99910   Mean   : 0.98752
 3rd Qu.: 2.03658   3rd Qu.: 2.01419
 Max.   : 5.03409   Max.   : 6.05318
```

```r
library(ggplot2)
```

```
Warning: il pacchetto 'ggplot2' è stato creato con R versione 4.3.3
```

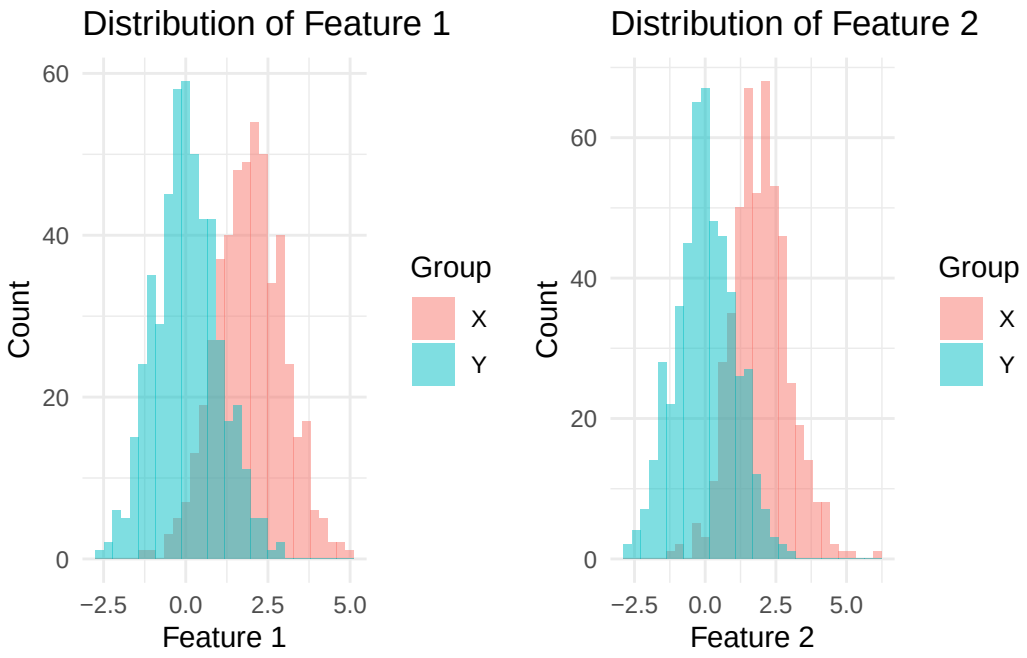
```r
ggplot(df, aes(x = Feature1, y = Feature2, color = Group)) +
  geom_point(alpha = 0.6, size = 2) +
  labs(title = "Scatter Plot of X and Y", x = "Feature 1", y = "Feature 2") +
  theme_minimal()
```



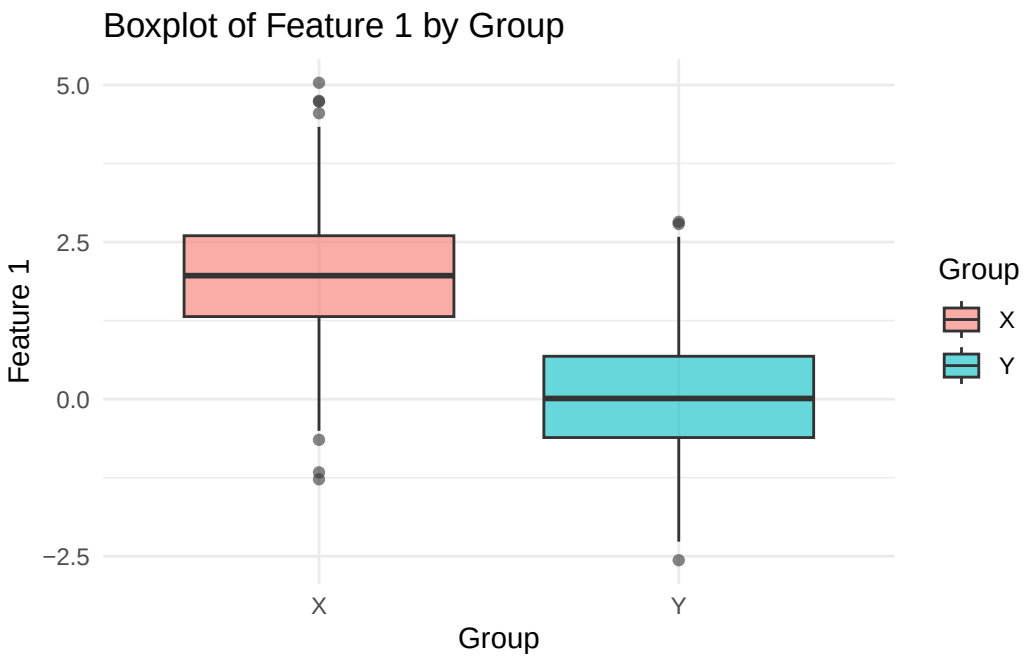
```r
#install.packages("ggpubr")
library(ggpubr)
```

```
Warning: il pacchetto 'ggpubr' è stato creato con R versione 4.3.3
```

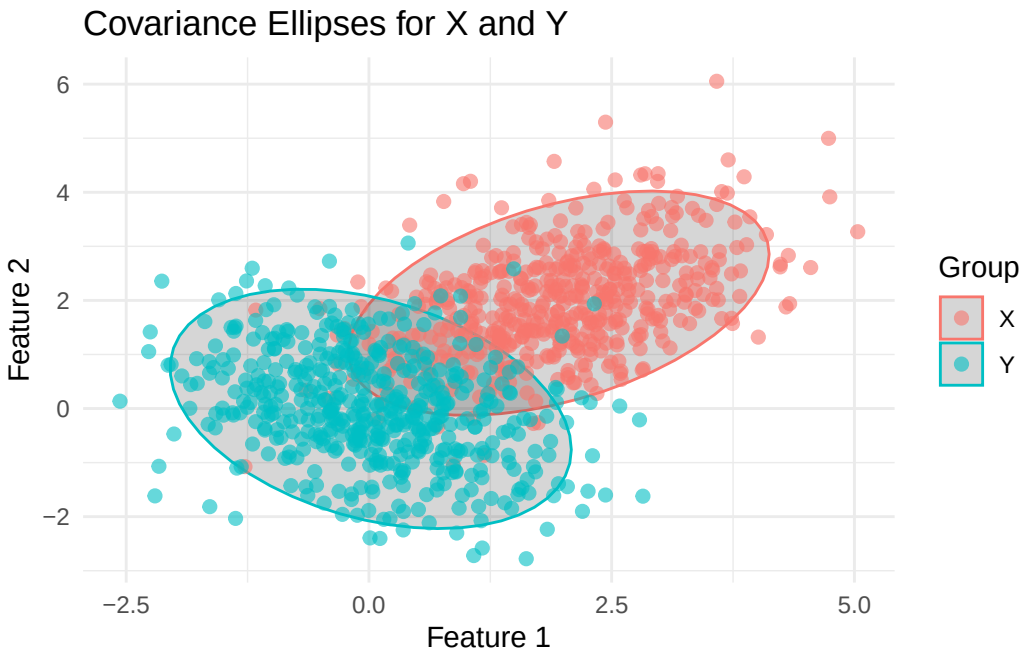
```r
p1 <- ggplot(df, aes(x = Feature1, fill = Group)) +
  geom_histogram(alpha = 0.5, position = "identity", bins = 30) +
  labs(title = "Distribution of Feature 1", x = "Feature 1", y = "Count") +
  theme_minimal()

p2 <- ggplot(df, aes(x = Feature2, fill = Group)) +
  geom_histogram(alpha = 0.5, position = "identity", bins = 30) +
  labs(title = "Distribution of Feature 2", x = "Feature 2", y = "Count") +
  theme_minimal()

ggarrange(p1, p2, ncol = 2, nrow = 1)
```




```r
ggplot(df, aes(x = Group, y = Feature1, fill = Group)) +
  geom_boxplot(alpha = 0.6) +
  labs(title = "Boxplot of Feature 1 by Group", y = "Feature 1") +
  theme_minimal()
```




```
ggplot(df, aes(x = Feature1, y = Feature2, color = Group)) +
  stat_ellipse(level = 0.95, geom = "polygon", alpha = 0.2) +
  geom_point(alpha = 0.6, size = 2) +
  labs(title = "Covariance Ellipses for X and Y", x = "Feature 1", y = "Feature 2") +
  theme_minimal()
```



Now we can start the main part

```
# Shuffle data to prevent order bias
set.seed(2) # my second lucky number
df = df[sample(nrow(df)), ]
head(df)
```

```
       Feature1   Feature2 Group
```

```
853  0.08580424 -0.5618636     Y
975  1.21129746  0.1574371     Y
710  0.93695814  1.2017654     Y
774 -0.17622187  1.2553929     Y
416  2.77846378  2.9532442     X
392  0.60958601  0.6372948     X
```

We train a Binary Classifier that learns to distinguish between X and Y.

So if Fx and Fy: - are different, the classifier will separate them - are similar, the classifier will perform poorly

```
library(caret)
```

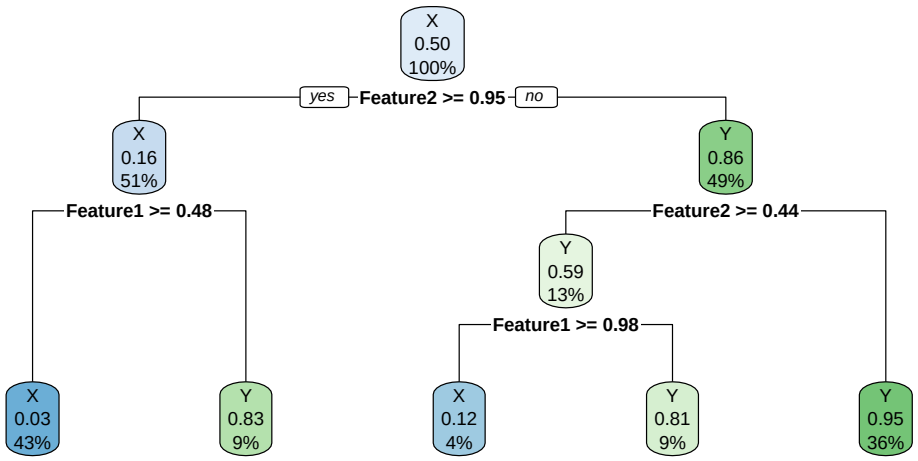
```
Caricamento del pacchetto richiesto: lattice
```

```
library(rpart)
# This time we avoid tuning the hyperparameters since it's only a PseudoCode or even better a
# Split 80/20
train_index = createDataPartition(df$Group, p = 0.8, list = FALSE)
train_data = df[train_index, ]
test_data = df[-train_index, ]

# CV
cv_control = trainControl(method = "cv", number = 5)

# Train
tree_classifier = train(Group ~ Feature1 + Feature2,
                        data = train_data,
                        method = "rpart",
                        trControl = cv_control,
                        tuneGrid = expand.grid(cp = 0.02)  # Increase complexity parameter,
)
```

```
library(rpart.plot)
```

```
Warning: il pacchetto 'rpart.plot' è stato creato con R versione 4.3.3
```

```
rpart.plot(tree_classifier$finalModel,
           main = "Decision Tree Splits (Cross-Validated)")
```



For each sample, the classifier assigns a probability score

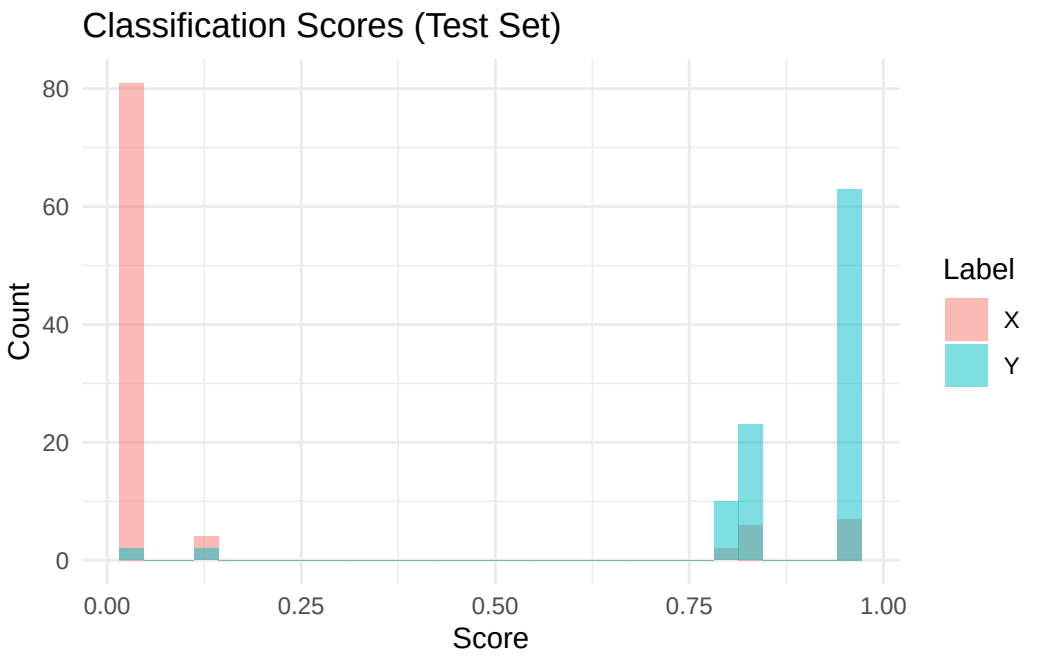
$$s_i = P(\text{sample } i \text{ belongs to class } 1)$$

```
test_data$Group = factor(test_data$Group, levels = c("X", "Y"))
test_probs = predict(tree_classifier, test_data, type = "prob")[,2]  # Probabilities for clas
```

Compute scores separately for X and Y:

$$S_X = \{s_1, s_2, \ldots, s_n\}, \quad S_Y = \{s_1, s_2, \ldots, s_m\}$$

```
scores_X = test_probs[test_data$Group == "X"]
scores_Y = test_probs[test_data$Group == "Y"]


score_df = data.frame(
  Scores = c(scores_X, scores_Y),
  Label = rep(c("X", "Y"), c(length(scores_X), length(scores_Y)))
)

# Plot
ggplot(score_df, aes(x = Scores, fill = Label)) +
  geom_histogram(position = "identity", alpha = 0.5, bins = 30) +
  labs(title = "Classification Scores (Test Set)", x = "Score", y = "Count") +
  theme_minimal()
```




```
library(pROC)
```

```
Warning: il pacchetto 'pROC' è stato creato con R versione 4.3.3
```

```
Type 'citation("pROC")' for a citation.
```

```

Caricamento pacchetto: 'pROC'
```

```
I seguenti oggetti sono mascherati da 'package:stats':
```

```

    cov, smooth, var
```

```
test_probs = predict(tree_classifier, test_data, type = "prob")[,2]
test_labels = as.numeric(test_data$Group) - 1
roc_obj = roc(test_labels, test_probs)
```

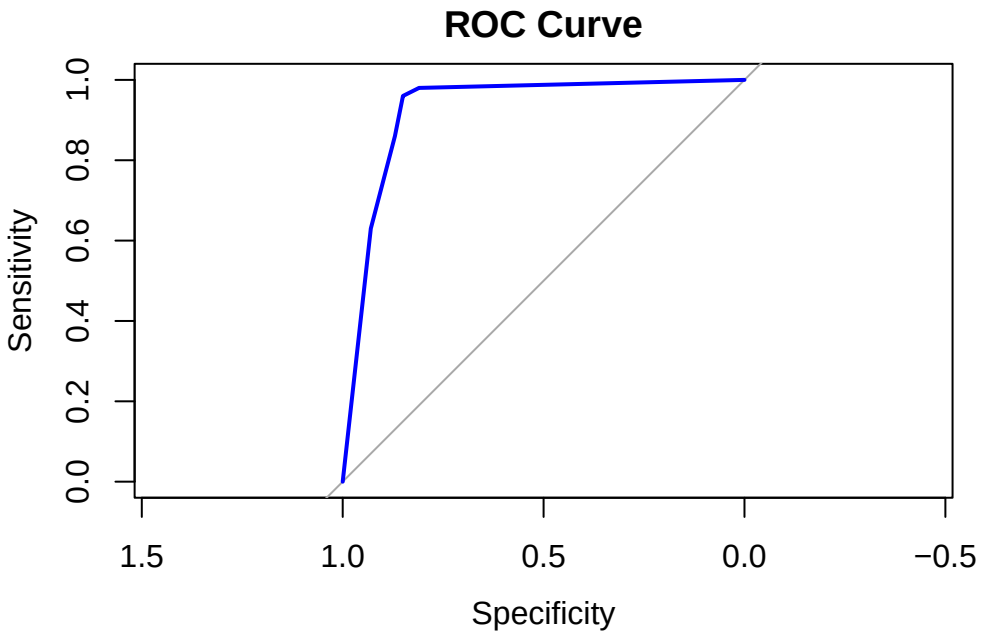
```
Setting levels: control = 0, case = 1
```

```
Setting direction: controls < cases
```

```
roc_obj
```

```

Call:
roc.default(response = test_labels, predictor = test_probs)

Data: test_probs in 100 controls (test_labels 0) < 100 cases (test_labels 1).
Area under the curve: 0.9256
```

```
#Plot
plot(roc_obj, col = "blue", lwd = 2, main = "ROC Curve")
```



Since we now have a single score per sample, apply a Univariate Two-Sample Test. The paper mention two methods:

Mann-Whitney U Test

$$U = n_X n_Y + \frac{n_X(n_X+1)}{2} - R_X$$

where: - R_X: sum of ranks assigned to X values

It compares medians using ranked data and is suitable when distributions are similar but shifted.

```
wilcox_test = wilcox.test(scores_X, scores_Y, exact = FALSE)
wilcox_test
```

```
    Wilcoxon rank sum test with continuity correction

data:  scores_X and scores_Y
W = 743.5, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

Edit: Printing the whole function and not the actual value allowed us to discover the real p value, I think R by default cut the value after a certain approximation so when I saw that both the values were 0 it was a little bit misleading, instead discovering the real value is reaaaally small but still a legit value was important.

Kolmogorov-Smirnov (KS) Test

$$D = \sup_x |F_X(x) - F_Y(x)|$$

where: - F_X(x) and F_Y(x) are empirical CDFs

Measures the maximum difference between cumulative distribution functions (CDFs). Detects broader distributional differences

```
ks_test = ks.test(scores_X, scores_Y, exact = FALSE)
```

```
Warning in ks.test.default(scores_X, scores_Y, exact = FALSE): il p-value sarà
approssimativo in presenza di legami
```

```
ks_test
```

```
    Asymptotic two-sample Kolmogorov-Smirnov test

data:  scores_X and scores_Y
D = 0.81, p-value < 2.2e-16
alternative hypothesis: two-sided
```

Avvertimento: il p-value sarà approssimativo in presenza di legami[1] We had this warning (sorry if it's italian), anyway exact = FALSE should deal with this issue (the test detected ties (identical values) in the data), it's caused by the simplicity of the dataset we choosed to analyze and consequentially the decision tree split assign identical scores to multiple data points (The decision tree provides discrete probability outputs)

```
alpha = 0.05  # Significance level

# Step 5: Decision Rule
if (wilcox_test$p.value < alpha) {
    print("Mann-Whitney U Test: Reject H0, distributions are significantly different")
} else {
    print("Mann-Whitney U Test: Fail to reject H0, no significant difference")
}
```

```
[1] "Mann-Whitney U Test: Reject H0, distributions are significantly different"
```

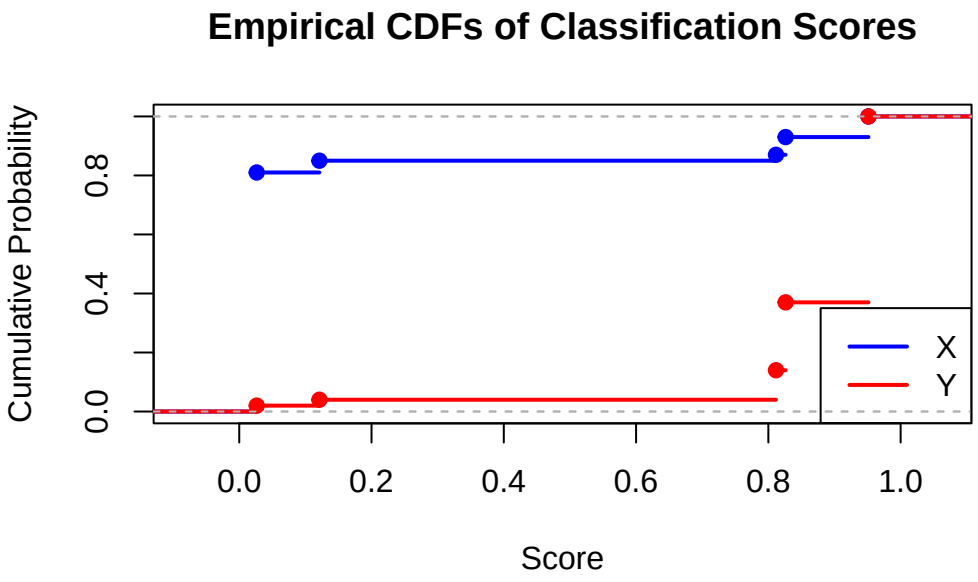
```
if (ks_test$p.value < alpha) {
    print("Kolmogorov-Smirnov Test: Reject H0, distributions are significantly different")
} else {
    print("Kolmogorov-Smirnov Test: Fail to reject H0, no significant difference")
}
```

```
[1] "Kolmogorov-Smirnov Test: Reject H0, distributions are significantly different"
```

Anyway both results indicate strong evidence against the null hypothesis (H0:X=Y), meaning that the two distributions are significantly different. Specifically Kolmogorov-Smirnov Test p-value: close to 0 indicates extreme separation between distributions The classifier assigns very different probability scores to X and Y meaning their empirical CDFs do not overlap at all In practical terms, the test detects a almost perfect separation.

Let's visualize it

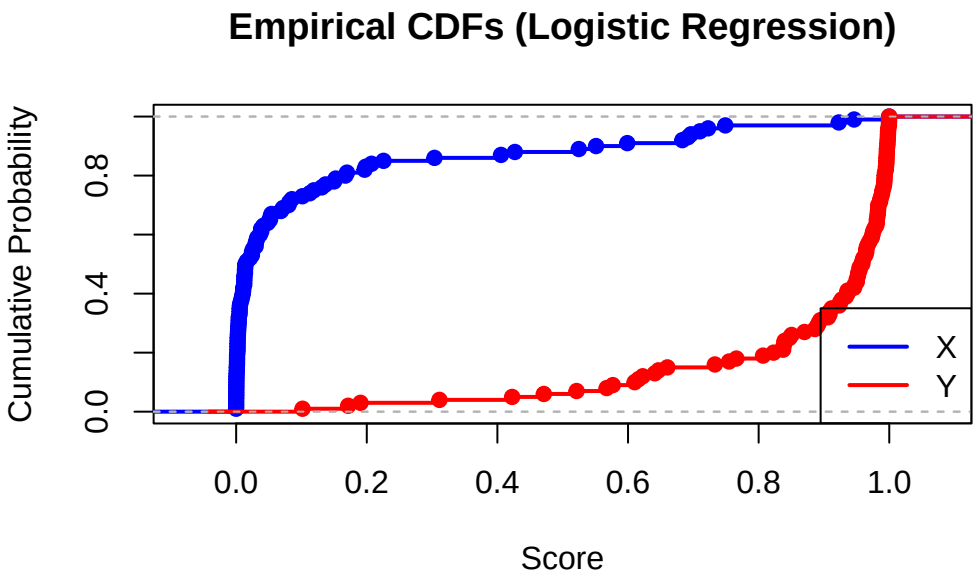
```
ecdf_X = ecdf(scores_X)
ecdf_Y = ecdf(scores_Y)

# Plot
plot(ecdf_X, col = "blue", lwd = 2, main = "Empirical CDFs of Classification Scores", xlab =
lines(ecdf_Y, col = "red", lwd = 2)
legend("bottomright", legend = c("X", "Y"), col = c("blue", "red"), lwd = 2)
```



Nonetheless the regularizion the decision tree is acting like a hard rule-based classifier, meaning it lacks generalization

Let's try to use a Less Deterministic Classifier like Logistic Regression to get softer probability estimates

```
logistic_model = train(Group ~ Feature1 + Feature2, data = train_data, method = "glm", family
test_probs_logistic = predict(logistic_model, test_data, type = "prob")[,2]

# Plot
ecdf_X_logistic = ecdf(test_probs_logistic[test_data$Group == "X"])
ecdf_Y_logistic = ecdf(test_probs_logistic[test_data$Group == "Y"])

plot(ecdf_X_logistic, col = "blue", lwd = 2, main = "Empirical CDFs (Logistic Regression)", x
lines(ecdf_Y_logistic, col = "red", lwd = 2)
legend("bottomright", legend = c("X", "Y"), col = c("blue", "red"), lwd = 2)
```



Logistic Regression provides a much more realistic probability distribution. We can see some natural overlap between X and Y making the tests valid since the classifier is no longer overconfident

```
# Mann-Whitney U Test
wilcox_test = wilcox.test(test_probs_logistic[test_data$Group == "X"],
                          test_probs_logistic[test_data$Group == "Y"],
                          exact = FALSE)
wilcox_test
```

```
    Wilcoxon rank sum test with continuity correction

data:  test_probs_logistic[test_data$Group == "X"] and test_probs_logistic[test_data$Group ==
W = 320, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

```
# Kolmogorov-Smirnov Test
ks_test = ks.test(test_probs_logistic[test_data$Group == "X"],
                  test_probs_logistic[test_data$Group == "Y"])
ks_test
```

```
    Asymptotic two-sample Kolmogorov-Smirnov test

data:  test_probs_logistic[test_data$Group == "X"] and test_probs_logistic[test_data$Group ==
D = 0.83, p-value < 2.2e-16
alternative hypothesis: two-sided
```

```
# Decision Rule
if (wilcox_test$p.value < alpha) {
    print("Mann-Whitney U Test: Reject H0, distributions are significantly different")
} else {
    print("Mann-Whitney U Test: Fail to reject H0, no significant difference")
}
```

```
[1] "Mann-Whitney U Test: Reject H0, distributions are significantly different"
```

```
if (ks_test$p.value < alpha) {
    print("Kolmogorov-Smirnov Test: Reject H0, distributions are significantly different")
} else {
    print("Kolmogorov-Smirnov Test: Fail to reject H0, no significant difference")
}
```

```
[1] "Kolmogorov-Smirnov Test: Reject H0, distributions are significantly different"
```

Even with logistic regression, the classifier separates X and Y extremely well. This suggests that our dataset is highly distinguishable, making most two-sample tests extremely sensitive.

```
roc_obj = roc(as.numeric(test_data$Group) - 1, test_probs_logistic)
```

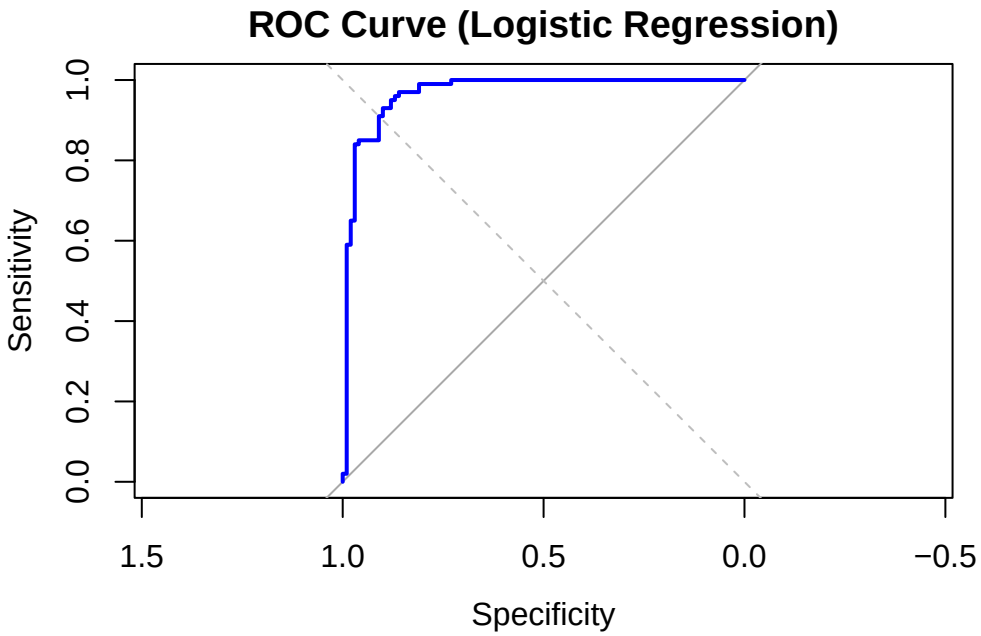
```
Setting levels: control = 0, case = 1
```

```
Setting direction: controls < cases
```

```
roc_obj
```

```
Call:
roc.default(response = as.numeric(test_data$Group) - 1, predictor = test_probs_logistic)

Data: test_probs_logistic in 100 controls (as.numeric(test_data$Group) - 1 0) < 100 cases (as
Area under the curve: 0.968
```

```r
plot(roc_obj, col = "blue", lwd = 2, main = "ROC Curve (Logistic Regression)")
abline(a = 0, b = 1, lty = 2, col = "gray")
```



AUC is close to 1 but not exactly 1, meaning the classifier is working well.

Friedman's Two-Sample Test - Pseudocode

Algorithm 1 Friedman's Two-Sample Test

Require: M (Number of Monte Carlo runs), n (Sample size per class), k (Feature dimension), μ (Mean shift), Σ (Covariance matrix), P (Number of permutations)

Initialize an empty list to store power values
for $m = 1$ to M **do** ▷ Monte Carlo Simulation
    Generate Class 0 samples: $X_0 \sim N(0, \Sigma)$
    Generate Class 1 samples: $X_1 \sim N(\mu, \Sigma)$
    Merge samples and assign labels:
    - $Y = 0$ for X_0
    - $Y = 1$ for X_1

    Train classifier $F(x)$ using logistic regression
    Compute test statistic T_{obs} using KS or MW test on classifier scores
    Initialize permutation null distribution list
    for $p = 1$ to P **do** ▷ Null Distribution Estimation
        Randomly shuffle labels Y
        Retrain classifier on shuffled data
        Compute test statistic T_p on permuted labels
        Store T_p in permutation null distribution
    end for
    Compute 95th percentile threshold:

$$T_{\text{null},95\%} = \text{quantile}(\{T_p\}, 0.95)$$

    Compute power:

$$\text{Power} = \frac{\sum_{m=1}^{M} (T_{\text{obs},m} > T_{\text{null},95\%})}{M}$$

    Store power estimate for current configuration
end for
Output results: Sample size n, Feature dimension k, Mean shift μ, Covariance structure, Computed power.

Simulating Power and Level

Two samples are pooled into a single dataset:

$$\{u_i\}_{i=1}^{N+M} = \{x_i\}_{i=1}^{N} \cup \{z_i\}_{i=1}^{M}$$

where: - xip(x) (sample from Null distribution) - ziq(x) (sample from Alternative distribution)

Observations originating from sample 1 are assigned yi=1, while those from sample 2 are assigned yi=0

$$y_i = \begin{cases} 1, & 1 \le i \le N \\ 0, & N+1 \le i \le N+M \end{cases}$$

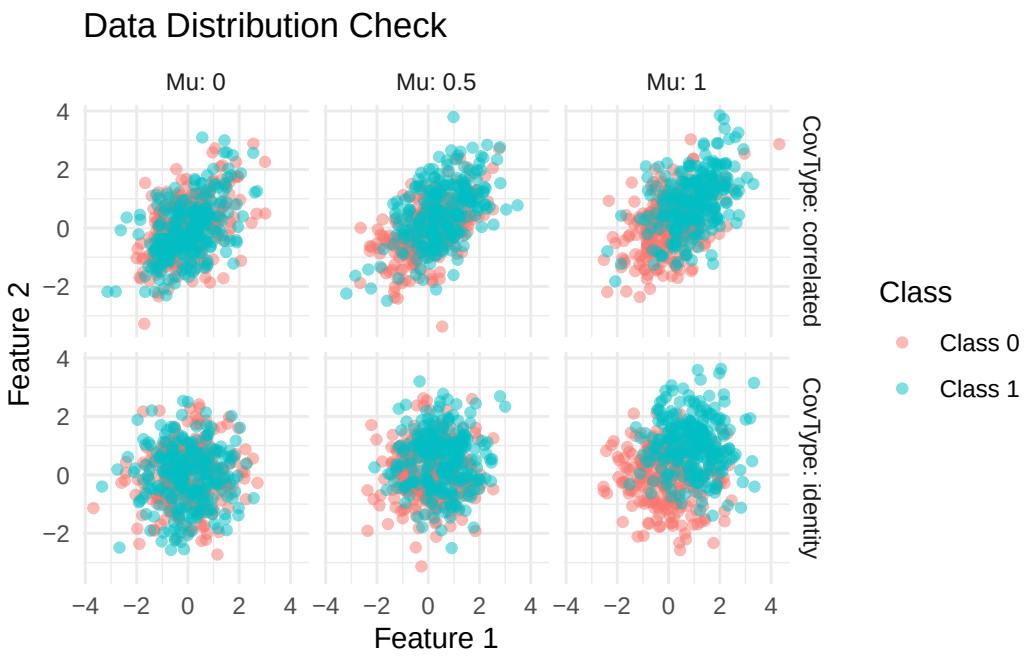
The problem ask how the power varies as we tweak the distance between F0 and F1. You may play around with k, n0 and n1 and possibly any other relevant quantity.

```
# Define parameters
n_values <- c(50, 100, 250)  #sample sizes
# We want to keep n as low as possible for efficiency:
# limited data to see if it struggle, a common size for balance and a larger one to see if re
# Expected effect? Power should increase as sample size grows, because larger datasets reduce

k_values <- c(2, 5, 10)  # feature dimensions
# A lower value also for visualization, a medium one to test a more realistic setting and a h
# Expected effect? Power should decrease as k increases, because higher dimensions add noise
```

```r
mean_shifts <- c(0, 0.5, 1)  # levels of separation
# with \mu = 0 distributions are identical, \mu = 0.5 tests a small difference between distri
# Expected effect? Power should increase as \mu increases, since larger separation improves c

cov_types <- c("identity", "correlated")  # Covariance types
# Identity matrix assumes features are independent, making computation easier while a Correla
# Expected effect? Correlated features may reduce power because classifiers may overfit to no
```

This values were "manually tuned" seeing the plot below: We want to keep the workflow as easy and efficient as possible and still provide a challenge to our classifier to allow us to get some results (using a clearly separated dataset would not provide us any valuable information right?)

```r
n_samples <- 250  # Number of samples per class for test
k <- 2  # for visualization

plot_data <- data.frame()

# I want to visualize how the scatterplot change depending on whatever covariance structure w
for (cov_type in cov_types) {
  for (mu in mean_shifts) {

    if (cov_type == "identity") {
      Sigma <- diag(k)  # Identity matrix
    } else {
      Sigma <- matrix(0.5, nrow = k, ncol = k)  # Correlated features
      diag(Sigma) <- 1
    }
    X0 <- mvrnorm(n_samples, mu = rep(0, k), Sigma = Sigma)  # Class 0
    X1 <- mvrnorm(n_samples, mu = rep(mu, k), Sigma = Sigma)  # Class 1
        df_0 <- data.frame(X1 = X0[, 1], X2 = X0[, 2], Class = "Class 0", CovType = cov_type,
    df_1 <- data.frame(X1 = X1[, 1], X2 = X1[, 2], Class = "Class 1", CovType = cov_type, Mu

    plot_data <- rbind(plot_data, df_0, df_1)
  }
}

plot_data$Class <- as.factor(plot_data$Class)
plot_data$CovType <- as.factor(plot_data$CovType)

# Plot
ggplot(plot_data, aes(x = X1, y = X2, color = Class)) +
  geom_point(alpha = 0.5) +
  facet_grid(CovType ~ Mu, labeller = label_both) +
  labs(title = "Data Distribution Check",
       x = "Feature 1", y = "Feature 2") +
  theme_minimal()
```



This is good enough, I focused on mu and the covariance structure since in previous test I saw that are really important into determine the power value: Moreover I HAD to use 2 dimension to allow a easy visualization while the value of n change mostly the proportion so focusing on those two variables was quite straighforward

Once defined our data we can go back to Friedman paper's.

A binary classifier F(u) is trained on the dataset to assign scores:

$$s_i = F(u_i), \quad i = 1, \ldots, N + M$$

$$S^+ = \{s_i\}_{i=1}^{N} \quad \text{(scores assigned to sample 1)} S^- = \{s_i\}_{i=N+1}^{N+M} \quad \text{(scores assigned to sample 2)}$$

Friedman suggests applying a univariate two-sample test (Kolmogorov-Smirnov or Mann-Whitney) to the classifier scores:

$$\hat{t} = T(S_+, S_-)$$

where T is a two-sample test statistic.

This part comes from Section 3.1 (Null Distribution):

The null hypothesis H0 :p(x)=q(x) is tested by computing the test statistic on permuted labels:

$$\{y_{j(i)}, u_i\}_{i=1}^{N+M}$$

where labels yi are randomly shuffled. This is repeated P times to construct the empirical null distribution

The power is computed as the proportion of times the observed test statistic exceeds the null distribution threshold:

$$\text{Power} = P(\hat{t} \geq \text{quantile}(\{\hat{t}_l\}_{l=1}^{P}, 1 - \alpha))$$

```
# Friedman's Two-Sample Test Function (Section 3)
simulate_friedman <- function(M, n, k, mu, Sigma) {

  power_values <- numeric(M)
  # Monte Carlo Simulation
  for (m in 1:M) {

    # The goal is to create a training dataset by pooling two samples, where
```

```
    X0 <- mvrnorm(n, mu = rep(0, k), Sigma = Sigma)  # Class 0 (Null)
    X1 <- mvrnorm(n, mu = rep(mu, k), Sigma = Sigma) # Class 1 (Shifted)

# Each observation is labeled as Class 0 (Null) or Class 1 (Shifted).
    data <- data.frame(rbind(X0, X1))
    labels <- c(rep(0, n), rep(1, n))
    data$Y <- factor(labels)

    # Train Logistic Regression Classifier to learn a scoring function F(x)
    # Friedman didn't specify any classifier, I think logistic is the easier choice
    model <- glm(Y ~ ., data = data, family = binomial())

    real_preds <- predict(model, data, type = "response")
    # Compute Kolmogorov-Smirnov Test Statistic
    observed_stat <- ks.test(real_preds[labels == 0], real_preds[labels == 1])$statistic

    # Number of permutations for null distribution
    P <- 100
    perm_values <- numeric(P)

    for (p in 1:P) {
      # Randomly shuffle labels
      perm_labels <- sample(labels)
      data$Y <- factor(perm_labels)
      # Same process as before
      perm_model <- glm(Y ~ ., data = data, family = binomial())
      fake_preds <- predict(perm_model, data, type = "response")
      perm_values[p] <- ks.test(fake_preds[perm_labels == 0], fake_preds[perm_labels == 1])$s
    }
    power_values[m] <- mean(quantile(perm_values, 0.95) <= observed_stat)
  }

  return(mean(power_values))
}
```

Simulations

```
M <- 100  # Number of Monte Carlo runs
results <- data.frame()

# for (n in n_values) {
#  for (k in k_values) {
#    for (mu in mean_shifts) {
#      for (cov_type in cov_types) {

#        Sigma <- if (cov_type == "identity") {
#          diag(k)  # Identity covariance
#        } else {
#          matrix(0.5, nrow = k, ncol = k) + diag(rep(0.5, k))  # Correlated covariance
#        }

#        power_value <- simulate_friedman(M, n, k, mu, Sigma)

#        results <- rbind(results, data.frame(
#          SampleSize = n, Dimension = k, MeanShift = mu, Covariance = cov_type, Power = powe
#        ))
```

```
        # Debug
#         cat("Completed:", "n =", n, "| k =", k, "| mu =", mu, "| Covariance =", cov_type, "|
#       }
#     }
#   }
#}
# print(results)

# write.csv(results, "friedman_test_results.csv", row.names = FALSE)
# Smarter move ever, I'll never run this chunck again in my life :)
# Anyway it took slighly less than 30 min, luckily when choosing the grid search I focused on
```

Avvertimento: glm.fit: si sono verificate probabilità stimate numericamente pari a 0 o 1 This was a common error when power = 1: Our classifier is so confident in our prediction that cause the dataset has a clear and complete sepration (this happened when mu=1)

Edit: you can't see the output cause when I was commenting the code for Quarto I accidentally rerun a line, as explained a few days ago above, I'm not gonna rerun the code ever again :)

```
# results <- read.csv("friedman_test_results.csv") # My saviour
# Quarto doesn't read CSV file apparently, so I'm gonna rewrite manually the whole df
# I tried with every LL;, no one was able to do this task, the one who got closer was DeepSee
results <- data.frame(
  SampleSize = c(50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 100,
  Dimension = c(2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10, 10, 2, 2, 2, 2, 2, 2,
  MeanShift = c(0, 0, 0.5, 0.5, 1, 1, 0, 0, 0.5, 0.5, 1, 1, 0, 0, 0.5, 0.5, 1, 1, 0, 0, 0.5,
  Covariance = c("identity", "correlated", "identity", "correlated", "identity", "correlated"
  Power = c(0.08, 0.07, 0.82, 0.67, 1, 0.99, 0.03, 0.04, 0.94, 0.58, 1, 1, 0.07, 0.09, 1, 0.4
)

print(results)
```

```
   SampleSize Dimension MeanShift Covariance Power
1          50         2       0.0   identity  0.08
2          50         2       0.0 correlated  0.07
3          50         2       0.5   identity  0.82
4          50         2       0.5 correlated  0.67
5          50         2       1.0   identity  1.00
6          50         2       1.0 correlated  0.99
7          50         5       0.0   identity  0.03
8          50         5       0.0 correlated  0.04
9          50         5       0.5   identity  0.94
10         50         5       0.5 correlated  0.58
11         50         5       1.0   identity  1.00
12         50         5       1.0 correlated  1.00
13         50        10       0.0   identity  0.07
14         50        10       0.0 correlated  0.09
15         50        10       0.5   identity  1.00
16         50        10       0.5 correlated  0.48
17         50        10       1.0   identity  1.00
18         50        10       1.0 correlated  0.99
19        100         2       0.0   identity  0.08
20        100         2       0.0 correlated  0.04
21        100         2       0.5   identity  0.98
22        100         2       0.5 correlated  0.85
23        100         2       1.0   identity  1.00
24        100         2       1.0 correlated  1.00
25        100         5       0.0   identity  0.04
```

```
26         100          5       0.0 correlated  0.03
27         100          5       0.5   identity  1.00
28         100          5       0.5 correlated  0.94
29         100          5       1.0   identity  1.00
30         100          5       1.0 correlated  1.00
31         100         10       0.0   identity  0.12
32         100         10       0.0 correlated  0.02
33         100         10       0.5   identity  1.00
34         100         10       0.5 correlated  0.87
35         100         10       1.0   identity  1.00
36         100         10       1.0 correlated  1.00
37         250          2       0.0   identity  0.01
38         250          2       0.0 correlated  0.03
39         250          2       0.5   identity  1.00
40         250          2       0.5 correlated  1.00
41         250          2       1.0   identity  1.00
42         250          2       1.0 correlated  1.00
43         250          5       0.0   identity  0.05
44         250          5       0.0 correlated  0.04
45         250          5       0.5   identity  1.00
46         250          5       0.5 correlated  1.00
47         250          5       1.0   identity  1.00
48         250          5       1.0 correlated  1.00
49         250         10       0.0   identity  0.04
50         250         10       0.0 correlated  0.06
51         250         10       0.5   identity  1.00
52         250         10       0.5 correlated  1.00
53         250         10       1.0   identity  1.00
54         250         10       1.0 correlated  1.00
```

To summarize the results into one plot:

```
library(dplyr)
```

```
Warning: il pacchetto 'dplyr' è stato creato con R versione 4.3.3
```

```

Caricamento pacchetto: 'dplyr'
```

```
Il seguente oggetto è mascherato da 'package:MASS':
```

```

    select
```

```
I seguenti oggetti sono mascherati da 'package:stats':
```

```

    filter, lag
```

```
I seguenti oggetti sono mascherati da 'package:base':
```

```

    intersect, setdiff, setequal, union
```

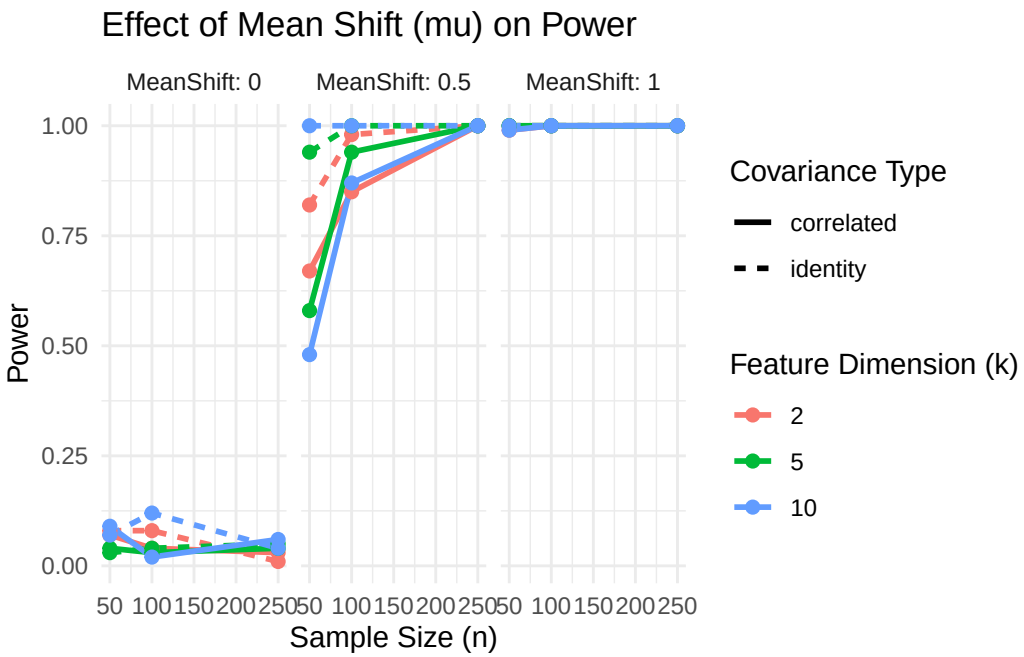
```
mu_effect <- results %>%
  group_by(SampleSize, MeanShift, Dimension, Covariance) %>%
  summarise(Power = mean(Power))
```

```
`summarise()` has grouped output by 'SampleSize', 'MeanShift', 'Dimension'. You
can override using the `.groups` argument.
```

```
ggplot(mu_effect, aes(x = SampleSize, y = Power, color = as.factor(Dimension), linetype = Cov
  geom_line(size = 1) +
  geom_point(size = 2) +
```

```
  facet_wrap(~MeanShift, labeller = label_both) +
  labs(title = "Effect of Mean Shift (mu) on Power",
       x = "Sample Size (n)", y = "Power",
       color = "Feature Dimension (k)",
       linetype = "Covariance Type") +
  theme_minimal()
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



Plot was good but we need to do better;

To answer the final question, so understand how each variable would influence the power results, the most logical think to do was to use the Partial dependence plots that is one of the main topics of Explanable AI. The problem is that we don't have access to a Real model, since we have runned Montecarlo simulation. PDP tecnically help visualize the marginal effect of a single variable(Each variable we iterated trough in our case) on the response (Power in our case) while averaging out the influence of other variables. Our solution to get a similar effect is to ì directly average results over other variables and plot marginal means; so we compute mean Power for each value

```
pdp_mu <- results %>%
  group_by(MeanShift) %>%
  summarize(Mean_Power = mean(Power))

pdp_n <- results %>%
  group_by(SampleSize) %>%
  summarize(Mean_Power = mean(Power))

pdp_k <- results %>%
  group_by(Dimension) %>%
  summarize(Mean_Power = mean(Power))

pdp_cov <- results %>%
  group_by(Covariance) %>%
  summarize(Mean_Power = mean(Power))
```

```
y_min <- min(results$Power)
y_max <- max(results$Power)

plot_pdp <- function(data, xvar, xlabel, title) {
  ggplot(data, aes_string(x = xvar, y = "Mean_Power")) +
    geom_line(size = 1.2, color = "blue") +
    geom_point(size = 3, color = "blue") +
    labs(title = title, x = xlabel, y = "Estimated Power") +
    ylim(y_min, y_max) +  # common axis
    theme_minimal()
}

plot_mu <- plot_pdp(pdp_mu, "MeanShift", "Mean Shift ()", "Effect of Mean Shift () on Power")
```

```
Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
i Please use tidy evaluation idioms with `aes()`.
i See also `vignette("ggplot2-in-packages")` for more information.
```

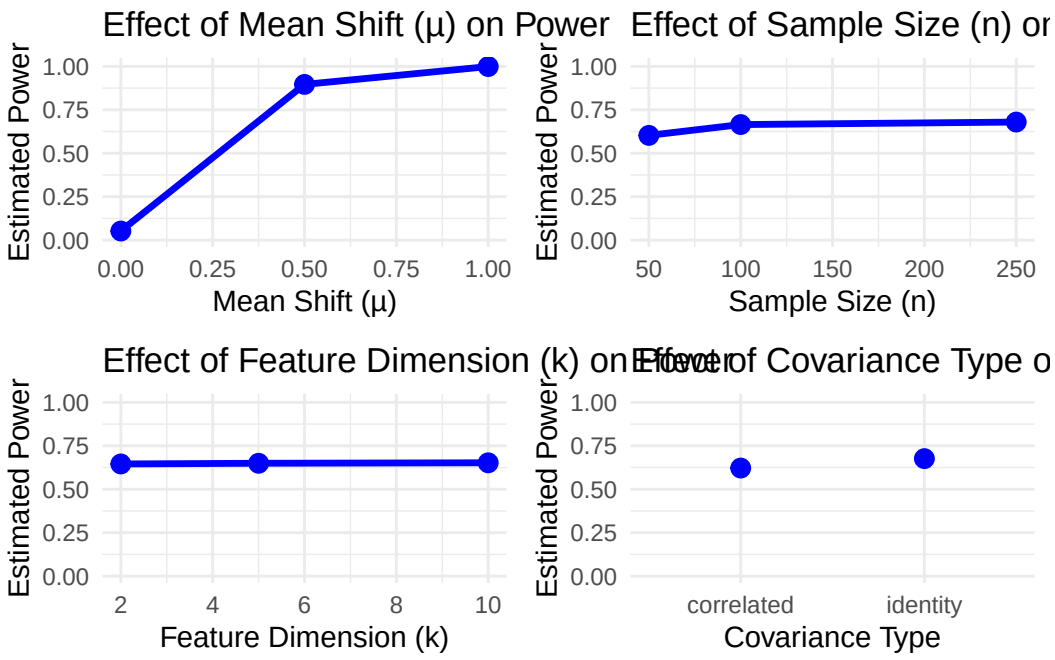
```
plot_n  <- plot_pdp(pdp_n, "SampleSize", "Sample Size (n)", "Effect of Sample Size (n) on Pow
plot_k  <- plot_pdp(pdp_k, "Dimension", "Feature Dimension (k)", "Effect of Feature Dimension
plot_cov <- plot_pdp(pdp_cov, "Covariance", "Covariance Type", "Effect of Covariance Type on
```

```
library(gridExtra)
```

```
Warning: il pacchetto 'gridExtra' è stato creato con R versione 4.3.3


Caricamento pacchetto: 'gridExtra'

Il seguente oggetto è mascherato da 'package:dplyr':

    combine
```

```
grid.arrange(plot_mu, plot_n, plot_k, plot_cov, ncol = 2)
```

```
`geom_line()`: Each group consists of only one observation.
i Do you need to adjust the group aesthetic?
```



- As expected, increasing leads to an increase in power and the separation ability of the classifier depends heavilty on this parameter.
- We can also say that the sample size matter more when dealing with limited data, as it grows we have a reduced effect that stabilize slightly after.
- The number of variables effect was quite surprising, I though that a higher number of variables would add complexity to the problem, instead the effect was quite flat.
- The covariance type have a smaller effect than I thought, logistic regression handles dependency well, so Correlated features do not drastically reduce power